Videojuegos

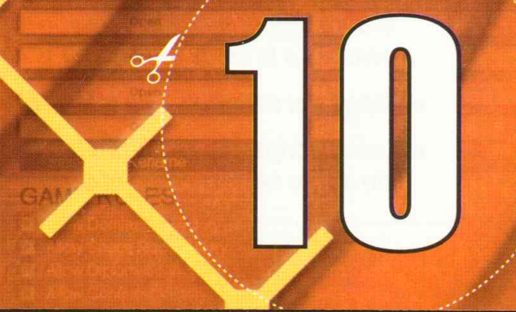
Curso de Diseño y Programación

Nº 10 5,99 euros

Análisis del **módulo principal** del juego

Manejo de objetos mesh en **Blitz 3D**

El modo multipista de **Cool Edit Pro**









Curso práctico de Videojuegos

AUTOR DE LA OBRA
Marcos Medina
DIRECCIÓN EDITORIAL
Eduardo Toribio
etoribio@iberprensa.com
COORDINACIÓN EDITORIAL
Eva-Margarita García
eva@iberprensa.com
DISEÑO Y MAQUETACIÓN
Antonio Gª Tomé
PRODUCCIÓN
Marisa Cogorro
SUSCRIPCIONES
Tel: 91 628 02 03
Fax: 91 628 09 35
suscripciones@iberprensa.com

FILMACIÓN: Fotpreim Duvial
IMPRESIÓN: Gráficas Don Bosco
DUPLICACIÓN CD.ROM: M.P.O.
DISTRIBUCIÓN
S.G.E.L.
Avda. Valdelaparra 29 (Pol. Ind.)
28108 Alcobendas (Madrid)
Tel.: 91 657 69 00

EDITA: Iberprensa
www.iberprensa.com


CONSEJERO
Carlos Peropadre

REDACCIÓN, PUBLICIDAD Y ADMINISTRACIÓN
C/ del Río Ter, 7 (Pol. Ind. "El Nogal")
28110 Algete (Madrid)
Tel.: 91 628 02 03
Fax: 91 628 09 35
(Añada 34 si llama desde fuera de España.)

DEPÓSITO LEGAL: M-35934-2002
ISBN: Coleccionable: 84 932417 2 5
Tomo 1: 84 932417 3 3
Obra Completa: 84 932417 5 X

NOTA IMPORTANTE:
Algunos programas incluidos en los CD de "Programación y Diseño de Videojuegos" son versiones completas, pero en otros casos se trata de versiones demo o trial, versiones de evaluación que Iberprensa quiere ofrecer a nuestros lectores. No se trata en ningún caso de las versiones comerciales de los programas, y las hemos incluido para dar al lector la oportunidad de conocer y probar esos programas y que así pueda decidir posteriormente si desea o no adquirir las versiones comerciales de cada uno.

Aprende divirtiéndote

Bienvenidos a **Programación y Diseño de Videojuegos**, la primera obra coleccionable cuyo objetivo es formar al alumno en las principales técnicas relacionadas en el desarrollo completo de un videojuego.

A lo largo de la obra el lector aprenderá programación a nivel general y a nivel específico con ciertas herramientas y lenguajes, aprenderá a trabajar con aplicaciones de retoque de imagen y también de diseño 3D y animación. Descubrirá las aplicaciones profesionales más importantes de audio y conocerá la historia de lo que se denomina "la industria del videojuego", los últimos 20 años, los juegos que marcaron un avance, sus creadores y en general la evolución del videojuego.

Pero además, esta obra tiene un segundo objetivo, desarrollar y potenciar la creatividad del lector, nosotros a lo largo de las diferentes entregas pondremos las bases y tú pondrás tu ingenio, tu creatividad y tu capacidad de mejorar.

Comienza aquí un viaje de 20 semanas articulado en 400 páginas y 20 CD-ROMs cuya finalidad es proporcionar las bases mínimas para después cada uno continuar su camino.

Recuerda que para alcanzar el éxito necesistas cumplir tres condiciones: que te gusten los juegos, poseer cierta dosis de creatividad y finalmente capacidad de estudio.

Una la cumples seguro.

sumario



PARA ENCUADERNAR LA OBRA:

- *Para encuadernar los dos volúmenes que componen la obra "Programación y Diseño de Videojuegos" se pondrán a la venta las tapas 1 y 2.*
- ***Tapas del volumen 1 ya a la venta.***
- ***Los suscriptores recibirán las tapas en su domicilio sin cargo alguno como obsequio de Iberprensa.***

SERVICIO TÉCNICO:
Para consultas, dudas técnicas y reclamaciones Iberprensa ofrece la siguiente dirección de correo electrónico: games@iberprensa.com

PETICIÓN DE NÚMEROS ATRASADOS:
El envío de números sueltos o atrasados se realizará contra reembolso del precio de venta al público más el coste de los gastos de envío. Pueden ser solicitados en el teléfono de atención al cliente 91 628 02 03

Módulo principal e indicadores de **pantalla**

Una vez estudiado cómo "Zone of Fighters" manipula el terreno de juego, le toca el turno a los procedimientos que nos permitirán mostrar todos los indicadores que aparecen durante el mismo. Pero antes vamos a analizar el módulo principal, el cual carga todos los demás módulos y gestiona el bucle principal del juego.

(■) EL MÓDULO PRINCIPAL

En todo juego hay una parte que gobierna el desarrollo de cada partida. Aprovechando la modularidad que nos brinda Blitz3D, vamos a liberar por medio de funciones el desarrollo de este bucle para obtener un código limpio, fácil de modificar y de corregir.

El módulo principal de nuestro juego se llama "zone_of_fighters.bb" y los primeros comandos que debemos escribir en él, y por lo tanto al principio de todo el código, son las definiciones (sobre todo si definimos constantes). Podemos tener todas las definiciones en un módulo aparte, así que debemos incluirlas en el módulo principal con la instrucción "Include". Ya que hemos diseñado un código modular, lo correcto será incluir todos los módulos más o menos por orden al principio del programa.



NOTA

Desde el módulo principal se llama a las demás funciones y es donde debemos compilar nuestro código.

Módulo presentación.bb
Función:
Presentación() : pantallas de presentación y argumento

Módulo menu.bb
Función:
Menu() : gestor principal del menú
Panel_opciones() : controla cada una de las opciones
Tabla_Records() : visualiza los records en el menú
Fondo_menu() : visualiza y actualiza el fondo 3D del menú

Módulo Funcarga.bb
Función:
Cargar_texturas_audio(): Carga imágenes, texturas y audio

1

Descripción de las funciones de los módulos: "presentacion.bb","menu.bb" y "Funcarga.bb".

```
Include "definiciones.bb"
Include "presentacion.bb"
Include "menu.bb"
Include "funcarga.bb"
Include "funcpantaudio.bb"
Include "decorado.bb"
Include "jugador_principal.bb"
Include "jugadores_CPU.bb"
Include "fjuego.bb"
Include "enemigos.bb"
```

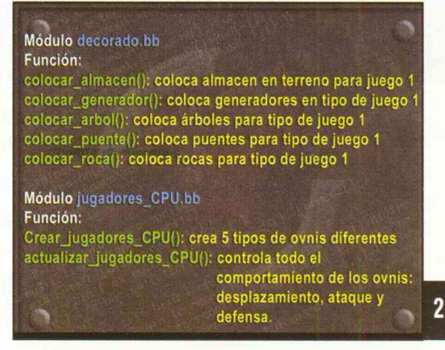
A partir de ese momento se incluirán en el programa principal todas las funciones contenidas en estos módulos.

El siguiente paso es mostrar la presentación de nuestro juego llamando a la función *Presentación()*. Inmediatamente después, llamamos a la función *Definir_Modo_Gráfico()* para establecer el modo gráfico que utilizará el juego a partir de ese momento.

Antes de cargar todas las texturas y sonidos y también antes de crear los modelos y los sprites que servirán de partículas, mostramos un aviso de la carga en la mitad de la pantalla.

Es siempre conveniente tener al usuario informado de todas las operaciones que realiza el programa en tiempos de espera, de esta forma el jugador sabrá si el programa sigue funcionando correctamente.

```
Locate (GraphicsWidth() Shr
1)-200,GraphicsHeight() Shr 1
```



Descripción de las funciones de los módulos: "decorado.bb" y "jugadores_CPU.bb".

```
Print "INICIALIZANDO ZONE OF
FIGTHERS... POR FAVOR ESPERE"
```

"Shr 1" es lo mismo que dividir entre 2. Es una manera más rápida y elegante de realizar una división. Con las funciones *GraphicsWidth()* y *GraphicsHeight()*, lo que hacemos es detectar el tamaño de la pantalla para poder imprimir la frase siempre en medio, independientemente de la resolución de ésta (Fig. 4).

Una cuestión importantísima a tener en cuenta es definir un rendimiento estándar para ordenadores de diferente velocidad, así que debemos crear una variable que guarde el número de fotogramas por segundo (Fps) máximo que pueda correr el juego. De esta forma, nos aseguramos de que en ordenadores superiores el



NOTA

Antes de utilizar cualquier función gráfica es imprescindible tener definido el modo gráfico. Además, cada vez que se defina un modo gráfico nuevo se borrarán de la memoria todas las texturas o modelos que hayamos cargado con anterioridad.

3

Módulo jugador_principal.bb
Función:
control_jugador_principal(): desplaza bionave y ejecuta las acciones del jugador
actualizar_jugador_principal(): actualiza el estado de la bionave en el juego
crear_disparo(): crea un disparo según el tipo elegido
actualizar_disparo(): desplaza el disparo, detecta colisiones y actualiza su estado
actualizar_agujero(): controla visualización marcas agüjeros
crear_explosion(): crea un explosión según tipo de disparo
actualizar_explosion(): controla visualizacióne explosiones
escudo_bionave(): controla activación del escudo
camuflaje_bionave(): controla activación del camuflaje
ocultar_bionave(): oculta bionave en vista de 1ª persona
mostrar_bionave(): visualiza bionave en vista 3ª persona
muere_jugador(): visualiza muerte del jugador

Descripción de las funciones del módulo "jugador_principal.bb".

juego no se ejecutará excesivamente rápido:

```
Timer = CreateTimer ( 60 ) ;
Velocidad máxima en frames
por segundo
```

A continuación, entramos en el bucle principal del juego, el cual contendrá otro bucle para cada partida. El bucle principal será infinito y es sólo a través del menú principal donde daremos la orden para salir del programa. Utilizaremos entonces el bucle "Repeat ...Forever".

Antes de entrar en el menú principal del juego vamos a inicializar el número de luchadores de cada nueva partida y la variable de salida principal "Escape". De vuelta del menú principal, ya tendremos definido, entre otras cosas, el tipo de juego y sabremos si es la primera partida o no (variable "Primer_juego") o si se trata de un juego nuevo (variable "Nuevo_juego"). El contenido de la variable "Primer_juego" es importante para el bucle principal, ya que será utilizada para impedir errores a la hora de llamar a las funciones de borrado antes de llamar a las de creación. La variable "Nuevo_juego" es utilizada para decirle al bucle principal que hemos vuelto del menú con una nueva partida creada o, simplemente, después de haber pulsado "Escape" durante el transcurso de una partida, por lo que impedimos que llame de nuevo a las funciones de creación. Si no lo hiciésemos así, cada vez que pulsemos "escape" durante el juego para ir al menú y "escape" de nuevo para volver al juego, se crearían de nuevo el entorno, los decorados y lo enemigos sobre los ya existentes, doblando así el número de polígonos.

```
Escape=0
numero_luchadores=0
Menu()
If Nuevo_juego=True
  If Primer_juego=False
    Borrar_Entorno()
    Borrar_Decorado()
    Borrar_Enemigos()
    ClearCollisions()
  EndIf
  Crear_Entorno()
  Crear_Decorado()
  Crear_jugadores_CPU(Numero_
  luchadores)
  Mapa_Colisiones()
  Nuevo_juego=False
  Primer_juego=False
EndIf
```

Si es un nuevo juego y es el primero: creamos el entorno, colocamos los decorados, creamos los jugadores enemigos según la cantidad elegida en el menú y almacenada en *Numero_luchadores* y creamos también el mapa de colisiones para todo el juego. A continuación, es importantísimo actualizar las variables "Nuevo_juego" y "Primer_juego" para indicarle al bucle principal la nueva situación.

Si es un nuevo juego y no es el primero (*Primer_juego=False*) debemos entonces borrar el entorno, los decorados, los enemigos y el mapa de colisiones para evitar duplicaciones antes de llegar a las funciones de creación.

Continuamos cargando la música que sonará al final de la partida ("musica2") para evitar tiempos de carga y la que sonará durante la partida, elegidas aleatoriamente de entre dos diferentes ("musica1").

```
musica2=LoadSound("c:\zone of
fighters\audio\musica3.wav")
m=Rnd(4,5)
musica1=LoadSound("c:\zone of
fighters\audio\musica"+Str(m)
+".wav")
c_musica1=PlaySound(musica1)
ChannelVolume c_musica1,
```



Con las funciones GraphicsWidth() y GraphicsHeight() es posible determinar el centro de la pantalla en cualquier resolución gráfica.

```
Volumen_Musica
```

Ya estamos preparados para entrar en el bucle que gobierna cada partida del juego. Empezamos estabilizando el rendimiento del juego para ordenadores de diferente velocidad con la instrucción "WaitTimer". Posteriormente detectamos si el jugador pulsa "escape", por lo que saldremos del juego y volveremos al menú. Seguidamente llamamos a las funciones que gobiernan el juego en sí. Actualizamos las colisiones y dibujamos cada escena. Seguidamente mostramos los indicadores de pantalla y el panel de ayuda antes de intercambiar los búferes.

```
Repeat
  WaitTimer(timer)
    If KeyDown(1) Escape=1
    Control_jugador_principal()
    Actualiza_Entorno()
    Actualizar_juego()
    Salvapantalla()
    Pausa()
    UpdateWorld
    RenderWorld
      Indicadores()
      If KeyDown(63)
        DrawImage control,
        (GraphicsWidth()/2)-
        (ImageWidth(control)/2),20
      EndIf
      Flip
  Until Escape=1
```

(■) INDICADORES DE PANTALLA

Una vez visto el funcionamiento del módulo principal del programa, pasaremos a explicar cómo se implementan los indicadores de



5

Según el tipo de disparo seleccionado se visualizará un punto de mira u otro.

pantalla. Vamos a crear una función que se encargue de esta tarea. Se llama *Indicadores()* y estará contenida en el módulo que actualiza todas las acciones del juego: "Fjuego.bb". Básicamente, lo que haremos en esta función es imprimir el estado de la puntuación, munición, vida, armamento, escudo y camuflaje, así como los puntos de mira que aparecen en el modo de primera persona.

Lo primero que haremos es implementar la impresión de estos puntos de mira (Fig. 5).

En función de las irregularidades del terreno la bionave subirá o bajará; por lo tanto, en primera persona la situación del punto de mira variará y no coincidirá con el centro real de la pantalla y como consecuencia no estará centrado con los disparos. Así que debemos variar la altura del punto de mira según esta variación en el terreno, para ello almacenaremos en la variable "y_puntomira" la posición vertical según la coordenada Y del terreno y siempre controlando que no baje de una altura de 100:

```
y_mira=(TerrainY(terreno1,
x,y,z))/4
y_puntomira=(GraphicsHeight()/
2)-y_mira
if y<100 y_puntomira=
y_puntomira+y_mira
```

Seguidamente, dependiendo del tipo de arma que tengamos seleccionada, dibujaremos un punto de mira u otro según "tipo_armamento" (Ver Código 1).

Seguidamente, simplemente definimos las coordenadas para los indicadores, siempre utilizando las funciones para averiguar el tamaño horizontal y vertical de la pantalla, ya que no sabemos en qué resolución se está jugando. De esta manera, podemos dibujar todos los indicadores siempre en el mismo lugar independientemente de la resolución escogida. (Ver Código 2).

Luego, colocamos las imágenes que contienen los gráficos de los indicadores y seguidamente imprimimos sus valores correspondientes. (Ver Código 3).

Todo el trabajo anterior se basa, sencillamente, en ajustar los gráficos y los valores en el lugar de la pantalla en donde se habían prefijado. Según el diseño del juego, debemos imprimir el estado de la vida de los enemigos (plantas, animales y ovnis) en la parte superior de la pantalla cada vez que reciban un impacto. Esta información se

Código 1. Punto de mira según armamento

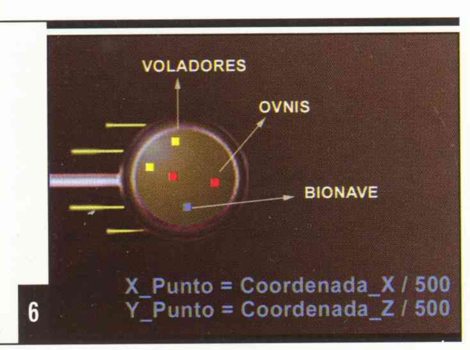
```
If tipo_camara=1 ; Primera persona
  Select tipo_armamento
   Case 1,2  DrawImage punto_mira,(GraphicsWidth()/2)-76,y_puntomira
   Default   DrawImage punto_mira3,(GraphicsWidth()/2)-30,y_puntomira
   End Select
EndIf
```

Código 2. Imágenes con gráficos de los indicadores

```
Xindicador_vidapuntos=GraphicsWidth()-(ImageWidth(indicador_vidapuntos)+20)
Yindicador_vidapuntos=GraphicsHeight()-ImageHeight(indicador_vidapuntos)
Xindicador_armamento=20
Xindicador_camuflaje=GraphicsWidth()-(ImageWidth(indicador_camuflaje))
Yindicador_armamento=GraphicsHeight()-ImageHeight(indicador_vidapuntos)
```

Código 3. Ajuste de los gráficos y los valores

```
DrawImage indicador_vidapuntos,Xindicador_vidapuntos,Yindicador_vidapuntos
DrawImage indicador_armamento,Xindicador_armamento,Yindicador_armamento
DrawImage indicador_escudo,0,Yindicador_armamento-40
DrawImage indicador_camuflaje,Xindicador_camuflaje,Yindicador_vidapuntos-40
Text Xindicador_vidapuntos+115,Yindicador_vidapuntos+22,puntos
Text Xindicador_vidapuntos+12,Yindicador_vidapuntos+20,vida
Text Xindicador_escudo+45,Yindicador_armamento-30,tiempo_escudo
Text Xindicador_camuflaje+40,Yindicador_vidapuntos-30,tiempo_camuflaje
Select tipo_armamento
   Case 1
     If municion<0 municion=0
     Text Xindicador_armamento+35,Yindicador_armamento+22,municion
     DrawImage disparo1B,Xindicador_armamento+154,Yindicador_armamento+16
   Case 2
     If municion2<0 municion2=0
     Text Xindicador_armamento+35,Yindicador_armamento+22,municion2
     DrawImage disparo2B,Xindicador_armamento+154,Yindicador_armamento+15
   Case 3
     If municion3<0 municion3=0
     Text Xindicador_armamento+35,Yindicador_armamento+22,municion3
     DrawImage disparo3B,Xindicador_armamento+151,Yindicador_armamento+20
   Default
     If municion4<0 municion4=0
     Text Xindicador_armamento+35,Yindicador_armamento+22,municion4
     DrawImage disparo4B,Xindicador_armamento+149,Yindicador_armamento+20
End Select
```



Esquema del radar del juego y la relación de las coordenadas del terreno (3D) con respecto a las del radar (2D).

visualizará en pantalla durante unos segundos y luego desaparecerá. Para realizar este procedimiento, debemos utilizar una variable que usaremos como contador "tiempo_en_pantalla". Esta variable la inicializamos a 50 cada vez que se produzca un impacto y se irá decrementando en uno cada vez que llamemos a la función de indicadores. De esta manera, mientras no valga 0 seguirá impresa en pantalla la información.

```
If sw_enpantalla=
1 And tiempo_en_pantalla>0
If informacion_vida<
0 informacion_vida=0
  Text (GraphicsWidth()/2),40,
informacion_vida
  tiempo_en_pantalla=
tiempo_en_pantalla-1
EndIf
```

Ya sólo nos falta mostrar el radar y actualizarlo con la función *Radar()* contenida en el módulo "Fjuego.bb".

```
DrawImage radar,0,40
Radar()
```

(■) FABRICANDO UN RADAR PARA EL JUEGO. FUNCIÓN RADAR()

El terreno de combate es amplio, por ello debemos facilitar al jugador la situación de los enemigos mediante la implementación de un sistema de seguimiento.

Se puede construir un radar de muchas formas diferentes. En "Zone of Fighters" utilizamos una serie de puntos de colores que simbolizan nuestra posición y la de los enemigos dentro del terreno de juego. De esta forma, los puntos de color rojo indican la posición de los ovnis, los amarillos, la de los voladores y el azul, la posición de la bionave protagonista.

Para poder representarlos en una pequeña porción de la pantalla (el tamaño del radar), vamos a utilizar las coordenadas de posición de cada uno de los objetos y trasladarlas al espacio del radar. Esto supone dividir el desplazamiento real de cada objeto unas 500 veces (siempre dependiendo del tamaño del radar), es decir: x/500, z/500

Esta división influirá también en la velocidad con que se moverán los puntos por el radar. Irán



Visión general de la posición de los indicadores de pantalla en el juego.

más rápidos, por ejemplo, con un valor inferior a 500 y más lentos con un valor superior. Hay que tener en cuenta también que si desplazamos los puntos más rápidos, el gráfico del radar debe ser mayor.

Es conveniente saber que la coordenada Z del plano 3D corresponde, en este caso, a la coordenada Y de la pantalla 2D, donde dibujaremos el radar (Fig. 6).

A continuación, debemos desplazar las coordenadas X e Y a la zona superior izquierda de la pantalla, que es donde tenemos situado el gráfico del radar. Así que empezamos dibujando los puntos amarillos que representan a cada uno de los 10 voladores que pueblan el terreno. Para ello, debemos recorrer con un bucle cada volador contenido en la estructura "tipo_voladores" (Ver Código 4).

Seguimos: con los puntos rojos que representan a los ovnis.

Y para finalizar, mostramos la posición del jugador. (Ver Código 5).

```
DrawImage punto_azul,
70-(x/500),50+(z/500)
```

Como hemos comprobado, implementar los indicadores de pantalla ha sido una tarea realmente sencilla (Fig. 7).

Código 4. Bucle que recorre cada volador

```
For volador.tipo_voladores= Each tipo_voladores
      CoordenadasX_volador= EntityX(volador\entidad_volador)
      CoordenadasZ_volador= EntityZ(volador\entidad_volador)
      DrawImage punto_amarillo,70-(Coordenadas_volador
/500),50+(CoordenadaZ_volador/500)
Next
```

Código 5. Representación de los ovnis

```
For jugadorCPU.tipo_jugadorCPU= Each tipo_jugadorCPU
      CoordenadasX_ovni= EntityX(jugadorCPU\entidad)
      CoordenadasZ_ovni= EntityZ(jugadorCPU\entidad)
      DrawImage punto_rojo,70-( CoordenadasX_ovni)/500),50+(
CoordenadasZ_ovni)/500)
Next
```

En el próximo número...

... implementaremos el código necesario para dotar a nuestra bionave de combate de movimiento.

Fabricando los elementos del **juego** (I)

Antes de empezar a fabricar los elementos del juego debemos terminar de animar la bionave de combate con Character FX.

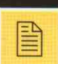
ANIMANDO LA CABEZA DEL PROTAGONISTA

Nos preparamos adecuadamente acomodando las vistas para poder realizar mejor el trabajo. Si es preciso, abrimos todas las ventanas que necesitemos. En el "Keyframer" pulsamos el botón para ir al primer frame y luego debemos crear el primer "Keyframe" pulsando en el botón para crearlo (Fig. 1).

Desplazamos la barra de navegación hasta colocar la línea de tiempo en el frame 15. Seguidamente, giramos la cabeza hacia la derecha, tal y como hicimos anteriormente, y pulsamos en el botón de "asignar Keyframe". Analizando la operación, hemos creado un punto inicial y otro final para el movimiento de la cabeza. Si reproducimos la animación, observaremos cómo desde el frame 1 hasta el 15 la cabeza gira suavemente hacia la derecha. Continuando con nuestra animación debemos realizar el giro de nuevo hacia el frente. Giramos de nuevo el eje Y y añadimos *Keyframer* (Fig. 2).

También podríamos copiar el primer frame al 30. Para hacer esto, debemos pulsar el botón para conmutar operaciones de copiado y desplazamiento para activar el modo de copiado (pulsado son operaciones de copiado y no pulsado de desplazamiento) Luego, seleccionamos, englobando con el ratón, toda la barra roja del frame 1. Ésta queda rodeada por un rectángulo amarillo. Posteriormente, nos situamos sobre este rectángulo y lo desplazamos hacia el frame 30 con el ratón. Observamos cómo el frame 1 se ha copiado en el 30.

A continuación, debemos grabar el giro hacia la izquierda, añadiendo más frames libres a la línea de tiempo y realizando las mismas operaciones anteriores (Fig. 3).



NOTA

Si nos quedamos sin frames libres en la línea de tiempo, podemos crear más, escribiendo el número de frames en la casilla de"frame final" y pulsando "Enter".



NOTA

Es importante saber que las diferentes animaciones deben ir unidas secuencialmente. De esta forma es posible extraerlas desde programación indicando los frames de comienzo y final de cada una.

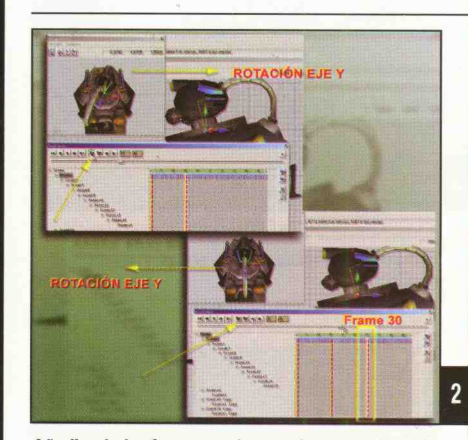
ANIMANDO LOS CAÑONES DE LA BIONAVE

El último paso de nuestra animación es crear un movimiento de retroceso para ambos cañones a la vez. Añadimos 10 frames más a la línea de tiempo, suficientes para realizar el retroceso y nos situamos en el frame 70.

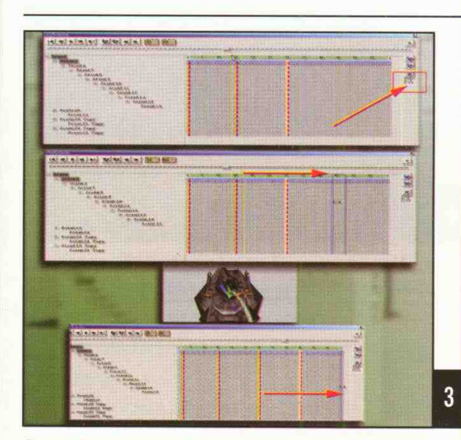
Seleccionamos el punto del extremo del cañón izquierdo y lo desplazamos hacia atrás un poco. A continuación, grabamos el "Keyframe". Nos situamos en el frame 75 y volvemos a desplazar el punto del extremo a su po-



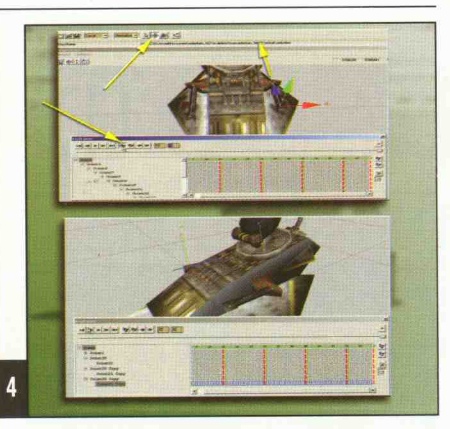
Procedimientos para empezar una animación en el modo "Animation".



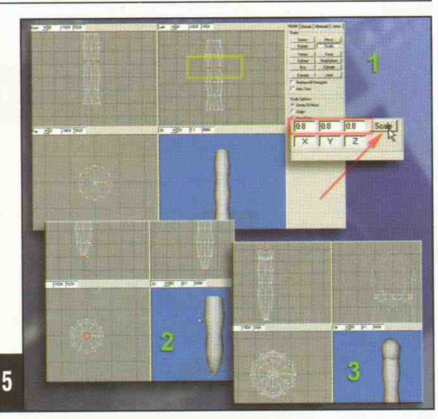
Añadiendo keyframes podemos almacenar los diferentes cambios en la animación.



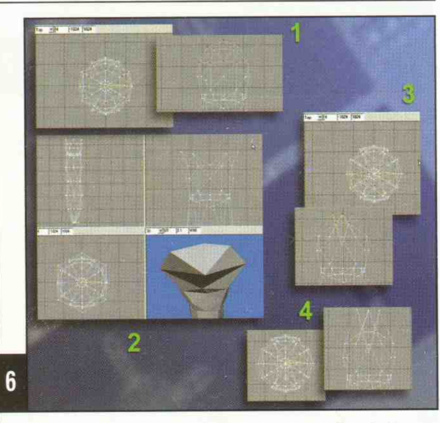
Es posible copiar y mover keyframes en la línea de tiempo de la animación.


4

En un mismo keyframe podemos añadir diferentes movimientos para los distintos esqueletos de un mismo modelo.


5

Procedimientos para el modelado del tronco del Slunk. Los anillos nos ayudarán a dar forma al tronco.


6

A partir de una geoesfera y desplazando vértices podemos obtener la forma de la boca.

sición original. De nuevo pulsamos el botón de grabación de "Keyframe" (Fig. 4).

Es posible añadir algunos movimientos de detalle más a la animación, por ejemplo, un pequeño giro hacia delante de la cabeza en el momento del disparo. Para ello, nos situamos en el frame 70 y giramos un poco la cabeza. Para añadir este nuevo movimiento sólo tenemos que pulsar de nuevo en el botón de grabación de "Keyframe". Desde el frame 65 al 70 la cabeza volverá automáticamente a su posición inicial, así que no es necesario volver a girarla.

GUARDANDO LA ANIMACIÓN

Una vez completada la animación de nuestro modelo, nos queda exportarla a disco para su utilización en Blitz3D. Esta operación la podemos realizar en varios formatos, desde .3DS y .X, hasta .MD2 y el formato estándar de Blitz3D .B3D.

Si disponemos de la versión comercial de Blitz3D 1.76 o superior, es conveniente grabar en formato .B3D por su manejabilidad posterior a la hora de implementar las animaciones mediante programación y por la total compatibilidad con todos lo métodos de colisión.

Si por el contrario trabajamos con versiones inferiores (versión demo), las cuales no soportan este nuevo formato, nos tendremos que conformar con utilizar el formato .MD2, muy manejable también, pero no compatible con todos los métodos de colisión.

Para exportar simplemente seleccionamos la opción *Exporter* del menú *File* y elegimos el formato adecuado, aceptando todos los parámetros por defecto.

FABRICANDO LOS ELEMENTOS DEL JUEGO

Continuamos nuestro desarrollo gráfico empezando la realización de los demás elementos del juego, desde los animales y plantas hasta todo el decorado. Para esta ocasión no utilizaremos la herramienta MercatorUV de Deep Paint 3D para realizar el mapeado UV de los modelos. En su lugar trabajaremos con la aplicación LithUnwarp, ya que precisaremos de un mapeado simple. Además, esta herramienta nos ayudará a optimizar de polígonos nuestros modelos. Una vez optimizado y mapeado en LithUnwrap, salvaremos el modelo 3D en formato .OBJ para luego pasar al Deep Paint 3D y texturizarlo. Pero antes debemos empezar por modelar cada uno de los elementos. En esta entrega realizaremos el gusano gigante *Slunk* y la planta *Luny*.

MODELADO DE UN GUSANO GIGANTE

CABEZA Y CUERPO

Ejecutamos MilkShape 3D y preparamos las vistas. La forma del cuerpo de los *Slunks* es básicamente cilíndrica, así que partiremos de un cilindro de 6 anillos (stack) y 12 divisiones (Slices). Cada una de las partes nos servirá para poder dar un poco de forma al cuerpo, así que, empezando por abajo, seleccionamos la fila segunda de vértices (con la opción "Select Backfaces" deseleccionada) y escalamos todos sus ejes con un valor de 0.8. Haremos lo mismo con las filas cuarta y sexta. Para crear la cola y así cerrar el cilindro, seleccionamos la primera fila de abajo y escalamos manualmente en la vista "top" hasta unir prácticamente todos los vértices entre sí. El siguiente paso es la cabeza, la cual haremos a partir de una geoesfera de densidad 1, es la que menos polígonos tiene y resulta ideal para modificar los vértices. Para finalizar, la ajustamos al tronco con la herramienta de escalado (F4) (Fig. 5).

Para dar la forma a la boca, seleccionamos las 3 filas de vértices superiores de la cabeza

y con la herramienta de escalado la abrimos. Seguidamente, seleccionamos el único vértice superior y lo bajamos como se muestra en la figura 6.

Siguiendo con la forma de la boca, seleccionamos los vértices que forman pico y los subimos para alargar la mandíbula. Posteriormente los seleccionamos y escalamos hacia dentro.

LAS PATAS

Vamos a añadir cuatro patas que situaremos en la base de la cabeza. Cada una de ellas está formada por dos cilindros. Uno de ellos de 1 "Stack" y 6 "Slices" y otro con 3 "Stack" y 6 "Slices". Al segundo cilindro le desplazamos los vértices de unión de cada división ("Stack") para obtener una forma más o menos curva. Para modelar el extremo de la pata, seleccionamos los tres vértices superiores (siempre con "Select Backfaces" deseleccionado) y los unimos con "Ctrl" + "N". Queremos obtener una forma más puntiaguda, así que desplazamos estos vértices: los inferiores para abajo, los del centro hacia fuera y los superiores hacia arriba (Fig. 7).

Terminamos, uniendo las dos partes de la pata por medio de sus vértices con "Ctrl" + "N" como se muestra en la figura 8 y colocándola en su lugar por debajo de las mandíbulas. Para realizar las siguientes patas, sólo es necesario hacer copias y aplicar procedimientos de espejo ("Mirror").

Seleccionamos las dos partes de la pata con la tecla "Shift" pulsada, a continuación la duplicamos con "Ctrl" + "D" y realizamos un "Mirror Back <-> Front". Para las otras dos patas, duplicamos, rotamos su eje Y en 90 grados y luego colocamos, duplicamos de nuevo y hacemos un "Mirror Left <-> Right"

Antes de guardar nuestro modelo debemos agrupar todas las partes para formar un solo objeto 3D. Para ello, las seleccionamos todas y en la pestaña *Groups* pulsamos el botón "Regroup".

Terminamos exportando en el formato de Milkshape 3D .MS3D en el directorio "C:\juego_ZOF\Slunk\Modelado" con el nombre "Slunk.ms3d".

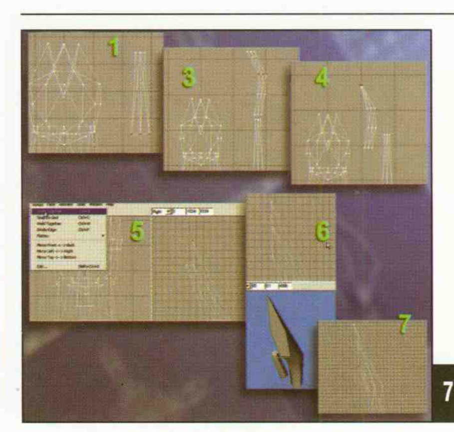
CREANDO EL MAPEADO UV CON LITHUNWRAP

Antes de texturizar el modelo, debemos generar un mapeado UV para que Deep Paint 3D pueda pintarlo correctamente. En vez del MercatorUV utilizaremos la herramienta LithUnWrap.

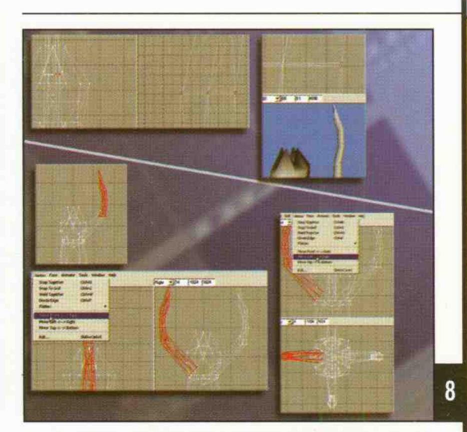
Este programa nos permite optimizar nuestros modelos y generar un mapeado UV del mismo. Una vez dentro de la aplicación cargamos el modelo "Slunk.ms3D" anterior en "File / Model / Open". Podemos observar en la pantalla un entramado de líneas, lo que significa que el objeto cargado no tiene aplicado ningún mapeado. Antes de generarlo nosotros con el programa, vamos a optimizar el modelo.
Seleccionamos todos los polígonos en *Tools / Select / All* (el entramado de líneas se volverá de color rojo), seguidamente elegimos la opción *Tools / Optimize models*. Una vez optimizado, procederemos a realizar el mapeado UV. Como el gusano tiene forma de tubo, es suficiente con generar un mapa lateral del mismo, para ello elegimos la opción *Tools / UV Mapping* y seleccionamos el modo "Planar", y aplicamos la opción "Y – Top" en "Positive". El entramado se transformará y mostrará una vista lateral de todo el *Slunk*. Para terminar exportamos de nuevo el modelo en *File / Model / Save* en formato .OBJ con el nombre *Slunk* en el mismo directorio para pasarlo al Deep Paint 3D. (Fig. 9).

TEXTURIZANDO CON DEEP PAINT 3D

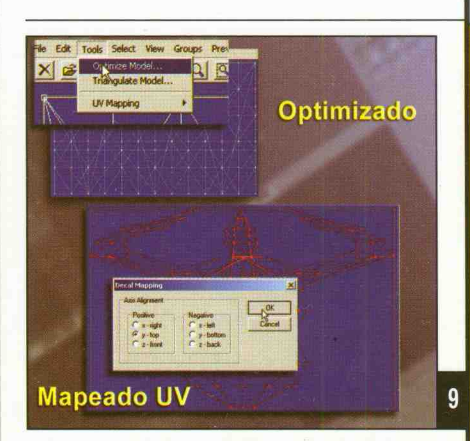
Al cargar nuestro modelo, observamos que en la ventana



A partir de cilindros es posible obtener cualquier forma desplazando vértices.



Copiando y realizando "mirrors" podemos crear y colocar el resto de las patas.

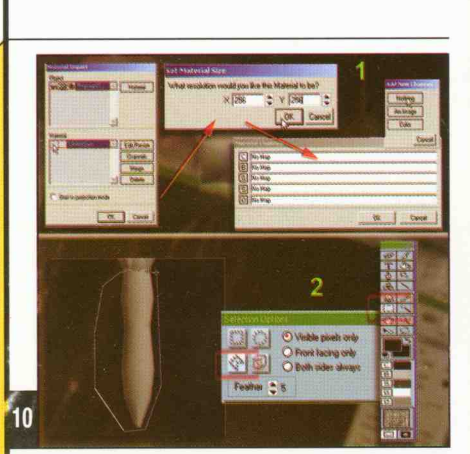


Con LithUnwrap podemos optimizar el modelo antes de crearle el Mapeado UV.

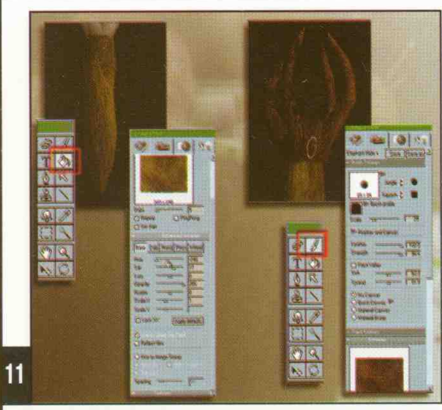


NOTA

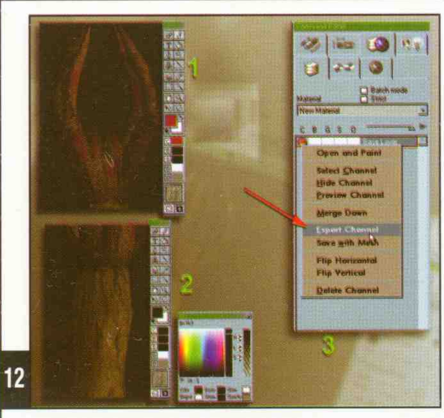
En el directorio "Extras" del CD se encuentran todos los modelos explicados en esta sección.



Podemos rellenar partes de un modelo seleccionándolo con anterioridad.



Mediante las herramientas de relleno y pincel podemos pintar nuestro modelo fácilmente.



Una vez acabado el texturizado podemos grabar la textura con la opción Export Channel en canal de color.



TRUCO

Podemos rotar el modelo de una manera más rápida, dejando pulsada la tecla "R" y desplazando el ratón con el botón izquierdo pulsado.

"Material Import" se indica la presencia de un mapeado UV. Debemos entonces crear un material vacío para poder continuar. Así que pinchamos en el primer cuadro de la sección "Material" en "Unknown" y elegimos un tamaño de 256 x 256 y como nuevo canal de color ("C") seleccionamos en "Add New Channel" "Nothing". Seguimos, seleccionando la vista "Top" en "View". Seguidamente ajustamos el modelo en la ventana y procedemos a pintarlo (Fig. 10).

Primeramente vamos a rellenar con una textura el cuerpo del gusano, para ello, lo seleccionamos con la herramienta de selección por líneas. Es necesario ir rotando el modelo y sumando selección, con la tecla "Shift" pulsada, para elegir todo las vistas del cuerpo.

Una vez seleccionado por completo el cuerpo elegimos la textura "Skin 3" (en la sección *Texture Paints* del *Command Panel*) y le modificamos el tono (Hue -248) y la saturación de color (Sat 77) en la pestaña *Base* en *Advanced Behaviour*. A continuación, rellenamos con la herramienta de relleno. Una vez pintado el cuerpo, pasaremos a la cabeza y patas. Eliminamos la selección anterior con "Ctrl" + "D" y elegimos la herramienta del pincel. Escogemos la textura "Elephant Hide +" en *Skins, Fur and Hair*. Luego, escalamos el pincel a un 25 % en la sección *Brush Settings* en "Scale" y pintamos sin pasar por encima del mismo sitio muchas veces (Fig.11).

Siempre es más profesional añadir detalles a la textura como manchas de sangre o cierta suciedad pegada en la piel. Para ello, sólo tenemos que pintar. Las machas de sangre las haremos con una variación del pincel ("Brush"). En "Variations" *(Command Panel)* elegimos la variedad "Pastel Oil +", la cual nos dará un trazo suave y diluido. Escalamos el pincel al 15%, seleccionamos una tonalidad de color rojo y pintamos sobre las mandíbulas y las patas, Para la suciedad elegimos la misma herramienta y variación de pincel, pero cambiamos a una tonalidad oscura y verdosa. Pintamos en la unión de la cabeza y el tronco y líneas sobre el tronco.

Una vez acabado, sólo nos queda exportar el material en el directorio "C:\juego_ZOF\slunk\texturizado\" con el nombre "textura_slunk.bmp". Para ello, elegimos la opción *Export Channel* del menú emergente que aparece al pulsar el botón derecho del ratón sobre el canal de color "C" en materiales.

MODELADO DE UNA PLANTA LUNY

El Modelado de un *Luny* es extremadamente sencillo. Sabemos que esta planta posee movimiento. Como éste lo aplicaremos en CharacterFx, tenemos que realizar el modelado en una postura rígida y erguida. Simplemente partimos de una esfera de 6 "Stacks" y 12 "Slices". Seleccionamos los vértices de la base, con *Ignore BackFaces* deseleccionado para escoger todo el perímetro, y los escalamos hacia fuera con F4 en la vista "Top". A continuación, hacemos lo mismo con los vértices centrales, hasta obtener la forma de una campana. Para terminar de dar forma debemos crear la boca, seleccionando las dos filas superiores de vértices y desplazándolas hacia el interior de la pieza.

Guardamos el modelo en el directorio "C:\juego_ZOF\modelado\luny\" con el nombre "luny.ms3d".

En el próximo número...

... terminaremos nuestro *Luny*, además de crear los *Shaarks*. También, seguiremos creando los elementos de "Zone of Fighters", en esta ocasión toca modelar y texturizar las plantas carnívoras.

Herramientas para hacer la música de un juego

Como ya sabemos, el complemento imprescindible para proporcionar audio a un juego es la música.

Hay multitud de sistemas para realizarla utilizando nuestro ordenador. Hasta ahora, hemos realizado efectos de sonidos aislados con editores de audio especializados como GoldWave, Sound Forge o Cool Edit Pro. Pero para hacer música necesitaremos algo más que simples efectos aislados. Vimos que la utilización de secuenciadores como Cubase VST nos permitía mezclar y sincronizar eventos musicales MIDI y muestras de audio para generar un tema musical.

UTILIZACIÓN DE CAJAS DE RITMOS

Existe otro tipo de secuenciadores muy utilizados hoy día que permiten unir muestras en forma de notas musicales, ritmos o voces para crear música. Son las cajas de ritmos virtuales. Actualmente, es muy común este tipo de aplicaciones para la creación de música de baile y juegos de ordenador ya que son de fácil uso y proporcionan un resultado excelente. Generalmente, son muy usadas para realizar ritmos o loops de muestras para incluir en un tema musical mayor, normalmente realizado en un secuenciador MIDI y a través de instrumentos externos. Hay infinidad de ellas, pero sólo explicaremos el uso básico de las dos cajas de ritmos más utilizadas y potentes: *Rebirth* y *Fruity Loops*. En ambas aplicaciones vamos a realizar un sencillo loop que posteriormente exportaremos en formato .WAV para poder insertarlo en nuestro secuenciador favorito.

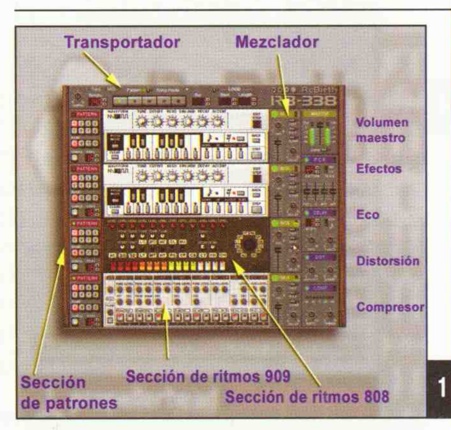
CREANDO UN LOOP CON "REBIRTH RB-338"

Rebirth- RB-338 nace de la idea de *Propellerhead* (Steinberg) de simular en un ordenador los módulos de sonido *TB 303, TR 808* y *TR 909* de *Roland*. Su manejo se basa en dos modos de trabajo: patrones y canciones. Los patrones nos servirán para crear canciones enteras, las cuales se podrán grabar en disco desde el compás indicado por "Start" hasta el indicado por "Lenght" con el modo "Loop" activado. En la figura 1 se muestra una descripción de los diferentes módulos de la interfaz del programa.

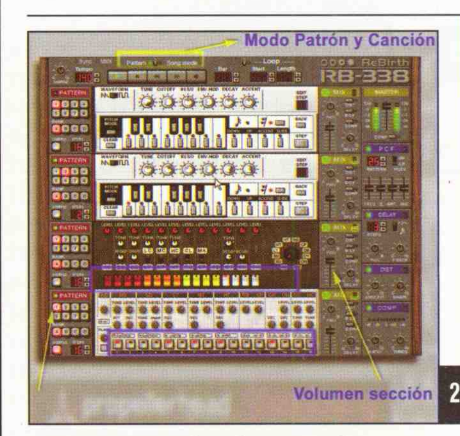
A continuación, vamos a construir un loop que luego exportaremos.

En primer lugar, crearemos patrones aprovechando las cuatro secciones de que disponemos. Trasladaremos estos patrones al modo "Song", en donde crearemos 12 compases para formar el loop definitivo. Una vez en el modo "Song" es posible modificar el patrón en tiempo real, cambiando parámetros o mutando secciones. Conmutamos a la sección "Pattern" para preparar nuestros patrones. Por defecto, un patrón está formado por 4 compases y un *tempo* de 140. Y en cada sección podemos disponer de 4 bancos (A,B,C,D) con 8 patrones cada uno. Así que en la sección de patrones pinchamos sobre el número 1 (patrón 1). La base rítmica la haremos con las secciones 808 y 909. Nos situamos en la sección 909 y hacemos clic sobre el primer botón de cada compás como se muestra en la figura 2.

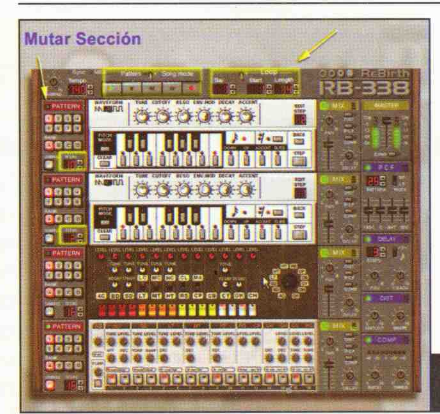
Para seguir la base rítmica añadiremos algunos golpes de percusión en la sección 808 y subimos el volumen de ésta en su mezclador. Una vez que tenemos



Descripción básica de las partes del programa Rebirth-338.



El primer paso es asignar las notas en las secciones de ritmos 808 y 909 en el modo "Pattern".



En el modo de grabación podemos activar o desactivar secciones, así como cambiar parámetros en tiempo real.

4

Descripción básica de las partes de Fruity Loops.



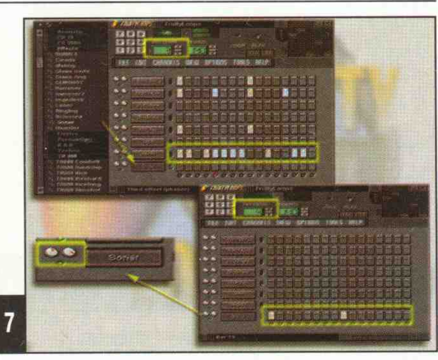
5

Es posible asignar cualquier muestra de las librerías de samplers a una pista (arriba). En la parte inferior se muestran las notas escritas en las pistas 1 y 2.



6

Podemos copiar patrones enteros con sólo las acciones de copiar y pegar.



7

Notas aplicadas a las pistas 5, 7 y 8.

la base rítmica, añadiremos los patrones de las secciones de sintetizador como se muestra igualmente en la figura 2.

Terminado de construir el patrón principal, pasamos al modo "Song", colocamos un 1 en "Start" y 14 en "Lenght". De esta forma creamos una canción de 14 compases. A continuación, debemos inicializar la canción con el nuevo patrón creado en el modo "Pattern". Para ello elegimos la opción *Edit / Initialize Song from Pattern Mode*. Mientras se graba, Rebirth permite modificar los parámetros, así como cambiar patrones o mutar las distintas secciones. Utilizaremos esta posibilidad para activar y desactivar las secciones a lo largo de los 14 compases. (Para tener una referencia, en "Extras" del CD se encuentra el loop terminado "loop_rebirth.wav"). Para finalizar, ya podemos exportar la canción como un loop con la opción *File / Export Loop as Audio File* (Fig. 3).

CREANDO UN LOOP CON FRUITY LOOPS

Fruity Loops es una estupenda herramienta para construir loops y canciones y exportarlas en formato .WAV o .MIDI. Posee un sintetizador de bajos integral, hasta 8 efectos simultáneos y además puede importar ficheros .WAV o .SYNC para crear librerías de sonidos. Vamos a explicar de una manera básica los pasos para crear un loop y exportarlo en formato .WAV.

En la figura 4 se muestra una pequeña descripción de las partes del programa.

Antes de comenzar a crear nuestros patrones, vamos a preparar los sonidos de cada pista. Por defecto, se visualizan un total de 8 pistas con 4 compases por cada una de ellas y un *tempo* de 140. Cambiemos el sonido de la primera pista. Para ello, abramos la librería de muestras de la "TR 808" haciendo clic sobre ella situada en la lista de samples. Sin dejar de pulsar el botón izquierdo del ratón, desplazar la muestra "TR808 Snr_snap" al botón de la pista primera. El sonido de esta pista cambiará. Pulsando el botón de cada pista podemos editar cada una de las muestras que contiene, abriéndose la ventana "Channel Settings". Haremos la misma operación para la pista 2, pero eligiendo la muestra "TR909 Klick" de la librería de la "TR 909" (Fig. 5).

Construyamos, a continuación, los patrones que nos servirán para formar el loop. Empezamos, pulsando el primer botón de cada compás en la segunda pista y el primero de los compases 1 y 3 de la primera. Seguidamente, crearemos otro patrón a partir del primero, pero añadiendo notas. Para ello, nos situamos en el primer patrón, lo seleccionamos por completo en *Edit / Select All* o pulsando "F3", y lo copiamos en memoria con "Ctrl" + "C". A continuación, seleccionamos el patrón 2 a través del selector de patrones o en la casilla "Pattern" y copiamos el patrón primero con "Ctrl" + "V". En el patrón 2 añadiremos más notas en las pistas 5 y 7 como se muestra en la figura 6.

Realizamos la misma operación anterior, copiando el patrón 2 en el 3 y añadiendo más notas en la pista 8, a la que previamente le hemos cambiado el sonido por la muestra "Sonar" de la librería "Effects". Además, a la pista 8 le subimos al máximo el volumen. Para terminar, creamos un cuarto patrón con sólo dos notas en la pista 8 al principio del compás primero y tercero. Bien, ya hemos construido los 4 patrones que utilizaremos para componer la canción que servirá de loop definitivo (Fig. 7).

Tanto Rebirth como Fruity Loops son aplicaciones que permiten un gran abanico de posibilidades para la creación de loops.

En el próximo número...

... terminaremos nuestro loops en Fruity Loops y seguiremos estudiando algunas características más de esta herramienta.

Manejo de funciones 3D (I). Objetos 3d

Para estudiar las funciones 3D que Blitz3D posee es necesario empezar por la creación y manejo de objetos 3D comúnmente denominados "mesh".

Como ya hemos visto a lo largo del curso, en Blitz3D es posible cargar y manipular *meshes* realizados con cualquier programa de modelado 3D. Pero también es posible crearlos, desde la nada, mediante programación y en tiempo real. Aprenderemos cómo utilizar ambos sistemas de una forma práctica y conoceremos los trucos que nos ayudarán a conseguir asombrosos efectos para nuestros juegos.

CREANDO PRIMITIVAS

Blitz3D permite la creación de objetos 3D básicos o primitivas. Podemos crear directamente: cubos, esferas, cilindros y conos. Para cada uno de ellos se utiliza un comando diferente:

cubo=CreateCube()
cilindro=CreateCylinder(16)
cono=CreateCone(16)
esfera=CreateSphere(8)
PositionEntity cubo,-5,0,8
PositionEntity cilindro,-1,0,8
PositionEntity cono,2,0,8
PositionEntity esfera,5,0,8

1 *Blitz3D permite la creación de primitivas básicas con una sola instrucción.*

■ **Cubo:**

```
CreateCube ([entidad madre])
```

■ **Esfera:**

```
CreateSphere([nº de segmentos]
[,entidad madre])
```

El nº de segmentos indicará más o menos polígonos para crear la esfera. Por defecto, su valor es 8; es decir, 224 polígonos, aunque es posible utilizar rangos desde 2 hasta 100.

■ **Cilindro:**

```
CreateCylinder([nº de segmentos]
[,sólido true/false]
[,entidad madre])
```

Si en la opción "sólido" colocamos "true" se creará un cilindro sólido y si colocamos "false" crearemos un tubo hueco. El rango posible en el nº de segmentos varía desde 3 hasta 100 inclusive.

■ **Cono:**

```
CreateCone(([nº de segmentos]
[,sólido true/false]
[,entidad madre])
```

Las opciones son exactamente iguales que en el comando anterior. La única diferencia es que el parámetro "sólido" indica si el cono tiene o no base (Ver "ejemplo1.bb").

CARGANDO OBJETOS 3D

Generalmente, necesitaremos utilizar en nuestros juegos objetos que hayamos creado previamente con cualquier herramienta de modelado. Blitz3D posee un par de comandos que posibilitan la importación de objetos desde un archivo, en formato .X, .3DS o .MD2 (además del formato .B3D a partir de la versión 1.76): "LoadMesh" y "LoadAnimMesh". Vamos a estudiar cómo usar cada uno de estos comandos, ya que es muy importante tener algunos conceptos claros si queremos manipular modelos formados por una sola parte, por varias o con animación.

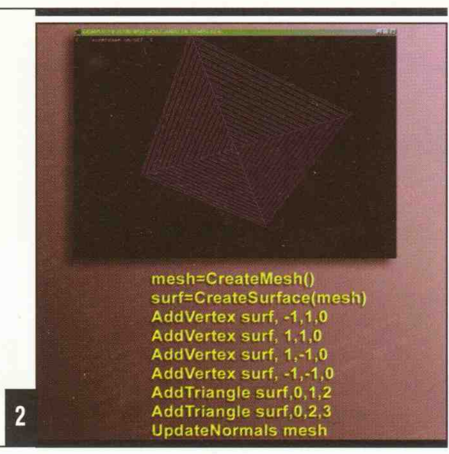
USO DE LOADMESH

Esta función nos permitirá importar un modelo 3D. El objeto cargado será convertido a un solo *mesh*; es decir, si cargamos, por ejemplo, el modelo de una persona, donde la cabeza, el tronco y las extremidades fueran partes diferentes, esta función no diferenciaría esas partes. Lo mismo ocurrirá si el modelo tuviese algún tipo de animación. De todas formas, tampoco resulta conveniente utilizar "LoadMesh" ya que también bajaría el rendimiento del programa. Así que esta función es ideal para cargar objetos estáticos formados por una sola parte, como una piedra o una pared.

```
LoadMesh (nombre fichero$
[,entidad madre])
```

LOADANIMMESH

Por otro lado, esta función nos permitirá importar modelos con algún tipo de animación. Además, es lo que debemos usar si queremos cargar un objeto que esté formado por varias partes, como por ejemplo, la cabeza, tronco y extremidades en el modelo de un hombre. Luego, mediante las funciones de manejo de "Children" ("CountChildren" o "GetChild")



Mediante programación es posible crear objetos 3D en tiempo real.

podremos texturizar cada parte o moverlas independientemente. Pero de estos procedimientos hablaremos después. Antes, vamos a continuar estudiando otros métodos de creación de *meshes*.

```
LoadAnimMesh (nombre fichero$
[,entidad madre])
```

CREANDO Y MODIFICANDO TUS PROPIOS *MESHES*. VÉRTICES, TRIÁNGULOS Y SUPERFICIES

Por medio de la programación es posible crear también cualquier tipo de objeto 3D más o menos simple; es decir, vamos a utilizar funciones de Blitz3D para modelar nuestros *meshes*. Pero antes, debemos aprender algunos conceptos básicos sobre cómo Blitz3D trabaja los objetos 3D.

Realmente, un *mesh* no está formado por vértices y triángulos, sino por superficies *(surfaces)*. Y son éstas las que están formadas por vértices y triángulos. Deducimos entonces que para tener una superficie es necesario al menos un triángulo y para formar un *mesh*, una superficie. Podemos, además, controlar el dibujado o renderizado de los triángulos de una superficie por medio de los "Brushes", con lo que podemos, por ejemplo, aplicar a cada triángulo propiedades como: pintar con un color, aplicar texturas o dar más brillo. Pero esta opción la veremos en el próximo número.

La forma que veremos para crear primitivas es similar a como lo haría una aplicación de modelado, la única diferencia es que ésta utiliza una interfaz gráfica y nosotros, programación. Vamos a crear un objeto 3D uniendo vértices y triángulos. Para empezar, es necesario crear un *mesh* nuevo con la función "CreateMesh". Para que este *mesh* tenga alguna forma es necesario crearle una superficie con la función "CreateSurface". A partir de ahora, ya estamos preparados para sumar vértices a esta superficie nueva con la función "AddVertex". Una vez creados los vértices debemos unirlos para formar los triángulos, los cuales darán la forma definitiva al *mesh*. Realizaremos esta operación con la función "AddTriangle". Para terminar el proceso debemos actualizar las normales del nuevo mesh, este procedimiento es imprescindible si queremos que se ilumine correctamente. Para ello aplicamos la función "UpdateNormals" (Ver "ejemplo2.bb").

Seguidamente, se muestran los pasos para crear el lado de un cuadrado utilizando estos procedimientos:

```
nuevo_Mesh=CreateMesh()
Superficie=CreateSurface(nuevo_Mesh)
AddVertex  Superficie, -1, 1, 0
AddVertex  Superficie, 1, 1, 0
AddVertex  Superficie, 1, -1, 0
AddVertex  Superficie, -1, -1, 0
AddTriangle Superficie, 0, 1, 2
AddTriangle Superficie, 0, 2, 3
UpdateNormals nuevo_Mesh
```

También es posible modificar nuestros propios *meshes* en tiempo real cambiando el color, la textura o la posición de sus vértices (Ver "ejemplo 3.bb").



Con "VertexCoords" podemos modificar las coordenadas de cada vértice posibilitando la deformación de superficies.

Si una vez creado el *mesh*, queremos modificar las coordenadas de los vértices de las superficies que lo componen debemos utilizar el comando "VertexCoords":

```
VertexCoords Superficie,
Nº del Vértice, Posición X#
del Vértice, Posición Y#,
Posición Z#
```

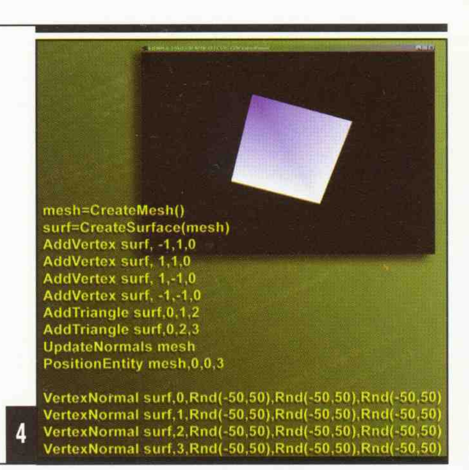
Si la superficie estuviera texturizada, también podemos cambiar las coordenadas de la misma mediante la instrucción "VertexTexCoords":

```
VertexTexCoords  Superficie,
Nº del Vértice, Coordenada U#
del vértice, Coord.. V#
[,coord.. W#]
```

U#, V# y W# son el equivalente a X#, Y# y Z# pero referidos al mapeado de textura.

Mediante la instrucción "VertexNormal" es posible cambiar la normal de un vértice; es decir, la dirección de éste con referencia a las coordenadas X, Y y Z:

```
VertexNormal Superficie,
Nº del Vértice, NX# (Normal
de la coordenada X del
vértice), NY#, NZ#
```



Modificando la normal de un vértice es posible cambiar su orientación con respecto al resto de los vértices de una superficie.

Por último, también es posible cambiar el color de cada vértice con la instrucción "VertexColor", por lo que conseguir superficies multicolores es tarea sencilla (Ver "ejemplo4.bb"):

```
VertexColor, Superficie,
Nº del Vértice, Componente
de Rojo#, Verde#, Azul#
```

Además, es posible pintar una superficie independiente con un determinado *brush* utilizando la función "PaintSurface":

```
PaintSurface Superficie,
Pincel (brush)
```

Es muy común, en muchos juegos, la utilización del sistema *LOD* (nivel de detalle) para la representación de los objetos 3D. Blitz3D sólo posee está técnica para la representación de terrenos, pero es posible realizar una función que lo haga también con los objetos. El fundamento principal de esta técnica es reducir polígonos; es decir, borrar superficies para luego crearlas de nuevo con menos vértices y triángulos, manteniendo siempre la forma original lo más aproximada posible. Para facilitarnos esta labor, Blitz3D nos suministra una función importante: "ClearSurface".

```
ClearSurface Superficie,
[Borrar todos los vértices
(true / false)][,Borrar todos
los triángulos (true / false]
```

Esta función nos permitirá borrar partes o superficies de un *mesh* inmediatamente.

MANIPULANDO OBJETOS 3D

Además de transformar en tiempo real cada vértice de la/s superficie/s de un *mesh*, podemos desplazar, rotar, escalar, pintar o iluminar un *mesh* completo, de igual forma como lo hacemos con una entidad. Para posicionar un *mesh* en el mundo 3D se utiliza el comando "PositionMesh":

```
PositionMesh Variable que
manipula el Mesh, X#, Y#, Z#
```

Aplicarle una rotación en cualquiera de sus 3 ejes es sencillo utilizando "RotateMesh":

```
RotateMesh Mesh, Pitch#(eje X),
Yaw# (eje Y), Roll# (eje Z)
```

También es posible cambiar el tamaño de cualquier objeto 3D con la instrucción "ScaleMesh":

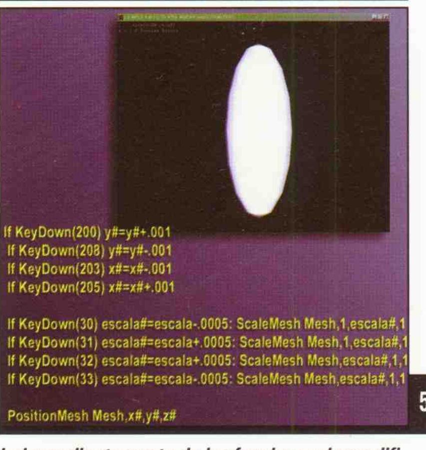
```
ScaleMesh Mesh, Escalado eje X#,
Escalado Y#, Escalado Z#
```

(Ver "ejemplo5.bb").

Si deseamos aplicar un escalado determinado uniforme a nuestro modelo debemos utilizar la instrucción "FitMesh". Ésta, además de escalar, realiza una traslación de los vértices del modelo:

```
FitMesh Mesh, Posicion X# del
mesh, Posición Y, Posición Z,
Longitud nueva del Mesh#,
Altura nueva del Mesh#,
Anchura nueva del Mesh#
[,Distorsion]
```

El parámetro de distorsión, por defecto en *false*, nos sirve para indicarle al Blitz3D que el *mesh* se ajustará uniformemente a la nueva medida; es



Independientemente de las funciones de modificación de entidades es posible utilizar funciones propias para los mesh como "PositionMesh".

decir, aumentar por igual el tamaño en los tres ejes.

También podemos ahorrarnos un montón de instrucciones si deseamos cambiar el aspecto de un *mesh*, ya que es posible pintarlo con un *brush* (pincel) previamente definido utilizando la instrucción "PaintMesh":

```
PaintMesh Mesh, Brush
```

Y por último, una interesante instrucción que nos permitirá iluminar un *mesh* completo o una porción de él; "LightMesh":

```
LightMesh Mesh, Componente de
color Rojo#, Verde#, Azul#
[,Rango de la luz], Posición X
de la luz#, Posición Y#,
Posición Z#
```

OBTENIENDO INFORMACIÓN DE LOS *MESHES*

Es muy común la necesidad de obtener cierta información de los modelos que utilizamos en nuestro juego como su tamaño, la cantidad de superficies que lo forman, etc.

Por ejemplo, queremos asignar un sistema de colisión por caja a un objeto 3D (ver sistema de colisión "EntityBox"). Para saber exactamente qué tamaño tendrá esa caja debemos saber el

tamaño del modelo. Para ello, disponemos de las instrucciones "MeshWidth", "MeshHeight" y "MeshDepth":

```
Longitud_Mesh = MeshWidth#
( Mesh )    ; X
Altura_Mesh = MeshHeight#
( Mesh )        ; Y
Anchura_Mesh = MeshDepth#
( Mesh )      ; Z
```

En ocasiones, puede ocurrir que necesitemos aislar una o varias superficies del conjunto que forma un *mesh*, para desplazarlas, cambiarlas de color, etc. Un buen ejemplo podría ser la implementación de daños en un coche de carreras, desplazando superficies de la estructura para crear abolladuras o rompiendo cristales. Existen un par de comandos que nos ayudarán a realizar esta operación: "CountSurfaces" y "GetSurfaces".

```
GetSurface (mesh, nº de superficie)
```

Mediante este comando obtenemos la superficie determinada por el parámetro "nº de superficie" en una variable, la cual servirá para manipular dicha superficie. Pero antes, es necesario recorrer en un bucle cada una de las superficies del modelo desde 1 hasta la última de ellas determinada por la función "CountSurfaces":

```
CountSurfaces (mesh)
```

Para terminar, una función realmente interesante y muy útil: "MeshesIntersect".

A veces, en muchos juegos se puede observar cómo parte del protagonista u otro personaje se introduce en la pared cuando se mueve. Este efecto queda realmente feo. Para evitar este problema podemos utilizar esta función. Lo que hace es simplemente avisar de cuándo dos objetos están mezclándose:

```
MeshesIntersect (Mesh A, Mesh B)
...
If MeshesIntersect ( Alien,
Pared) = True    Text "El
Alien se mezcla con la pared"
...
```

No es conveniente su utilización masiva ya que no es muy rápida y puede bajar el rendimiento del juego (Ver "Ejemplo6.bb").

6

```
Mesh=LoadMesh("bionave.3ds")
Pared=CreateCube()
ScaleMesh Pared,2,20,50
PositionEntity Pared,25,-10,90

Repeat
TurnEntity Mesh,1,1,1
RenderWorld
If MeshesIntersect(Mesh,Pared)=True
  Text 20,20,"BIONAVE INTRODUCIDA EN LA PARED !!"
Endif
Flip
Until Keydown(1)
```

La función "MeshesIntersect" nos permite controlar si dos modelos se están mezclando.

UTILIZACIÓN DE "CHILD"

Supongamos que hemos modelado un hombre con Milkshape 3D y salvamos el modelo sin agrupar, ya que la cabeza, el tronco y las extremidades son partes diferentes. Al no agrupar, el modelo estará formado por varios *meshes*. Estas partes son tratadas en Blitz3D como hijos (Children). Esto nos ayudará si queremos texturizar o mover cada parte independientemente. Además, cada una de ellas puede ser tratada como una entidad diferente con todo el poder que ello supone.

Podemos saber cuántas partes tiene un modelo contando los hijos del mismo. Para ello utilizaremos la función *CountChildren (Entidad)*



Mediante el manejo de "Children" es posible controlar las partes que forman un modelo.

Incluso si hemos nombrado cada parte en MilkShape 3D (u otra aplicación), es posible preguntar desde Blitz3D por ese nombre para encontrar esa parte (Child) utilizando la función:

```
FindChild (Entidad, nombre
de la parte$)
```

También podemos obtener una parte por su posición en la jerarquía del modelo mediante la función *GetChild (Entidad, número de la parte)*.

Si deseamos, por ejemplo, modificar la posición de todas las partes de un modelo, debemos utilizar un bucle que recorra cada una de ellas (Ver Fig. 7):

```
Hombre = LoadAnimMesh ("hombre.3ds")
For n = 1 to CountChildren (Hombre)
   Parte = GetChild (Hombre , n)
   MoveEntity Parte, 1, 1, 1
Next
```

Hasta aquí nuestro primer estudio de las funciones 3D para el manejo de *meshes*.



En el próximo número...

... nos dedicaremos al texturizado de objetos y a explicar los *brushes*.

Editores de audio. Cool Edit Pro 2.0 (II)

Vamos a terminar esta serie de tutoriales sobre editores de audio aprendiendo a trabajar con el multipista de Cool Edit Pro 2.

Podemos observar que el *modo multipista* es básicamente igual al *modo de edición*, la diferencia radica en que es posible trabajar con varios sonidos a la vez y mezclarlos. La mejor forma de aprender es practicando, así que vamos a realizar la mezcla de audio para crear el MP3 que sirve de argumento para nuestro juego "Zone of Fighters".

PREPARANDO LOS SONIDOS ANTES DE MEZCLAR

En el directorio "Extras" del CD disponemos de dos ficheros .WAV . Uno de ellos corresponde a la música que utilizaremos de fondo y el otro es una grabación en bruto de una voz en off narrando el argumento del juego. Nuestra misión es preparar esa voz para luego mezclarla con la música en el multipista.

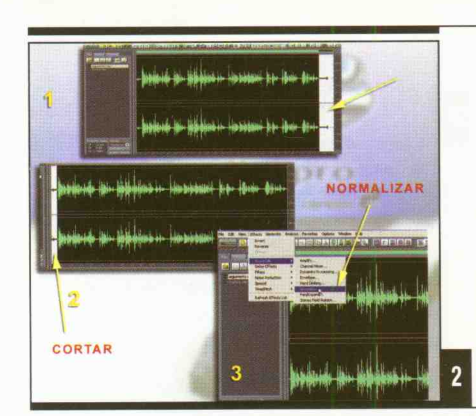
Una vez cargado el programa, nos encontramos en la ventana principal de edición, la cual explicamos en el número anterior de este tutorial. Pulsando en el icono pasamos al *modo multipista*. Allí vamos a insertar en la primera pista el audio correspondiente a la música que irá de fondo. Nos situamos sobre la primera pista vacía y pulsamos el botón derecho del ratón. Aparecerá un menú flotante en el que elegiremos la opción *Insert /Wave from File*. Elegimos el fichero contenido en el directorio "Extras" del CD llamado "musica.wav". Inmeditamente, aparece un bloque de audio en la pista primera conteniendo la música (Fig. 1).

Podemos insertar ficheros de audio directamente en las pistas en la ventana multitrack.

Vamos a continuación a insertar la voz en la segunda pista. Así que pulsamos sobre ella. Pero antes de seguir, es preciso preparar la voz en el *modo edición*, ya que está en bruto. Pulsamos en el botón para pasar al *modo edición*. Una vez situados en él cargamos del CD el fichero llamado "argumento.wav", el cual vamos a preparar para mezclar posteriormente con la música. Una vez cargado, vamos a eliminar el audio sobrante del principio y fin, seleccionando las partes a borrar y pulsando "Ctrl" + "X" o pulsando la tecla "Suprimir". A continuación, y como viene siendo habitual, realizaremos un normalizado del volumen al 100 % mediante la opción *Effects / Amplitude/Normalize* (Fig. 2).

Seguidamente, aplicaremos un efecto de reverberación o eco para simular la inmensidad del espacio exterior. Para ello, elegimos el efecto "Reverbs" situado en *Effects/Delay effects*. Elegimos el preset *Concert Hall Light* y cambiamos los siguientes parámetros del efecto:

- **Total Reverb Length** (Longitud total del reverb) a 1190 ms.
- **Attack Time** (Tiempo de ataque) a 240 ms.



El primer proceso es cortar el audio sobrante y normalizar el volumen.

- **High Frequency Absorption Time** (Tiempo en eliminar las altas frecuencias) a 630 ms.
- **Perception** (cantidad percepción, con suavidad o eco) a 30.

A continuación, también cambiamos los parámetros de mezcla del efecto:

- **Original Signal (Dry)** (Porcentaje de señal original) al 70 %.
- **Reverb (Wet)** (Cantidad de reverberación) un 165 %.

MEZCLANDO TODO EN EL MULTIPISTA

Una vez aplicado el efecto, ya tenemos la voz preparada para pasarla al multipista y mezclarla con la música. Debemos entonces insertar el sonido a la ventana *Multitrack*, eligiendo la opción *Insert in Multitrack* en *Edit* o pulsando "Ctrl" + "M" (Fig. 3). De nuevo, pasamos al modo multipista y observamos cómo en la pista segunda se ha insertado correctamente un nuevo bloque conteniendo el audio de la voz ya preparado.

Para desplazar el bloque de la voz a lo largo de la pista, por ejemplo, para ajustar su reproducción en un determinado lugar, nos situamos sobre ella y move-



Una vez terminado el tratamiento del sonido podemos insertarlo en el multipista con "CTRL" + "M".

mos el ratón sin dejar de pulsar el botón derecho. Una vez situados los bloques de audio debemos ajustar los volúmenes de cada pista para conseguir una mezcla perfecta.

En primer lugar, vamos realizar una subida rápida de volumen al comienzo del bloque de voz, para que su entrada en escena se produzca de forma suave. Para ello, debemos pulsar en el botón para visualizar en el bloque la gráfica de la envolvente del sonido, es decir, el volumen. En un principio, se muestran dos cuadrados blancos, al principio y al final de la onda, unidos por una línea. Esta línea representa el nivel de volumen del sonido. Pulsando sobre cada cuadrado podemos desplazarlos y así cambiar el volumen en ese punto. Para realizar, por ejemplo, subidas o bajadas de volumen, podemos crear todos los puntos (cuadrados) de inflexión que queramos con sólo pulsar con el ratón sobre la línea. De momento, nos interesa una subida rápida al principio. Solamente tenemos que crear un punto nuevo a continuación del primero. Este punto indicará el tope del volumen en la subida y el primero el comienzo.

Hay una forma más rápida de aumentar el volumen general de una pista y es a través del vúmetro que aparece al pulsar con el botón derecho sobre "V0" □. Vamos a continuar preparando los cambios de volumen en la pista de la música. Para ello, realizaremos la misma operación anterior, pero en esta ocasión añadiremos más puntos de inflexión para crear diferentes cambios de volumen dependiendo de la zona que esté sonando. Bajaremos entonces el volumen cuando esté reproduciéndose la voz (para mantener de fondo la música) y lo subiremos cuando ésta haya acabado.

Podemos también añadir cambios en el panorámico del sonido de la misma forma que hicimos con el volumen, a través de una gráfica y puntos de inflexión. Para visualizar esta gráfica pulsamos sobre el botón . A continuación, realizaremos los cambios en el panorámico sumando puntos de inflexión y desplazándolos.

También podemos cambiar el panorámico de una pista de manera general pulsando sobre "Pan0" con el botón derecho del ratón.

OTRAS OPCIONES

A través del multipista podemos cambiar los parámetros de un efecto aplicado a un sonido insertado en una pista. Además, también es posible añadirle efectos nuevos.

Para editar un efecto, es imprescindible que el sonido ya tenga uno asignado desde el modo *edición*. En nuestro caso, recordemos que en el audio de la voz en off habíamos aplicado una reverberación. Pues bien, para editar desde el multipista ese efecto, sólo tenemos que pulsar sobre el botón "FX2" . Aparecerá entonces un panel gráfico con todos los parámetros del efecto. Además, podemos mezclar si queremos que suene más o menos reverberación pulsando en la pestaña "Mixer" en ese mismo panel.

CONVIRTIENDO TODAS LAS PISTAS EN UN SOLO FICHERO DE AUDIO

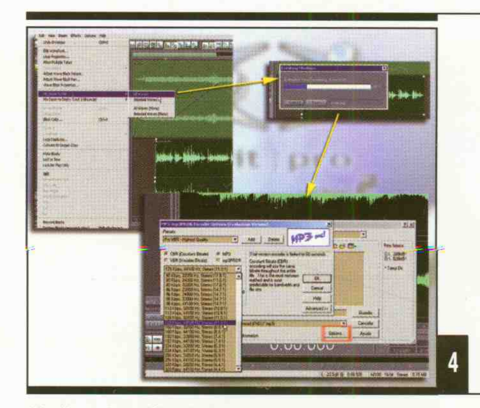
Una vez terminada la mezcla, necesitaremos obtener un solo fichero de audio para que pueda ser incluido en nuestro juego. Debemos entonces elegir la opción *Mix Down to File* en el menú *Edit* del multipista y elegir la opción *All waves*. Con esto, le estamos diciendo a Cool Edit Pro 2 que nos mezcle todas las pistas de audio en una sola. Cuando el programa termina el proceso de mezclado, pasa directamente a la ventana de edición y nos muestra la onda resultante del proceso. (Fig. 4).

Seguidamente, podemos normalizar de nuevo el volumen para tener definitivamente preparado el argumento del juego listo para grabar en formato .MP3.

GRABANDO EN .MP3

Para salvar el audio en MP3 elegimos la opción *File / Save as*. En el apartado *Tipo* elegimos el formato *mp3PRO(Fhg)(*.mp3)*. Después, elegimos la calidad resultante pulsando en el botón "Options" y elegiendo la opción *128 Kbps, 44100 Hz, Stereo (11.0:1)*.

En este número hemos conocido la enorme utilidad del *modo multipista* de Cool Edit Pro 2. Ponemos fin aquí a la serie de tutoriales que hemos dedicado a los editores de audio profesionales más utilizados.



Mediante la gráfica de la envolvente podemos modificar el volumen de la pista en cualquier lugar.



... comenzamos una serie dedicada al desarrollo y programación para juegos. Empezaremos por los mundos BSP en Blitz3D.

Juegos de estrategia (I). Estrategia por **turnos**

Empezamos una serie de dos números dedicados a los juegos de estrategia. Miraremos atrás en la Historia para conocer los antecedentes que han servido para crear la estrategia en nuestro PC.

La clasificación de este género se reduce, básicamente, a juegos de estrategia por turnos y estrategia en tiempo real.

DEL TABLERO AL ORDENADOR

La estrategia siempre ha estado ligada al arte de la guerra. Este tipo de juegos ha existido a lo largo de la Historia en forma de tableros. Quizás el más popular y antiguo sea el ajedrez. Pero como estrategia bélica jugada sobre un tablero fueron *Estratego* y *Warhammer* los que ganaron popularidad a partir de los años 40. El primero se basaba en las guerras en Europa, mientras que el segundo se basaba en mundos de fantasía repletas de batallas entre humanos y orcos. Éstos y otros juegos de tablero tenían un denominador común: el uso de los dados para determinar el rumbo de una partida.
Con la llegada de los ordenadores personales, todos estos títulos, que al principio llenaban una mesa entera de fichas, cartas y figuras en miniatura de las tropas, pasaron a la pantalla. El azar de los dados se transforma en decisiones lógicas del ordenador tomadas de una gran base de datos o analizando cientos de factores en un segundo.

En este tipo de juegos, y al igual que ocurre en una partida de ajedrez o damas, el jugador debe plantear la siguiente acción a realizar, mientras el contrario (ordenador o humano en un sistema multijugador) se limita a esperar los resultados del movimiento.

Este tipo de estrategia tiene infinidad de adeptos en todo el mundo y, hoy día, existen muchos títulos que, a pesar del enorme éxito del género en tiempo real, siguen ocupando un lugar importante en el mercado.

LOS PRIMEROS JUEGOS Y LA EVOLUCIÓN TÉCNICA

Realmente, debemos empezar por aquellos títulos que convirtieron los juegos de tablero al ordenador en la década de los 90. Entre ellos destacamos *Warhammer: Shadow of the Horned Rat* (Mindscape, 1996), fiel al juego de tablero más popular en el mundo de la estrategia. Antes de *Warhammer* aparecieron otros títulos ambientados en la segunda guerra mundial como *Steel Panthers* (MindScape, 1995) o la serie de campañas *Camping Series* de Talonsoft. No olvidemos que, al margen de la fantasía y las guerras mundiales, se encontraban

NOTA

MEZCLANDO TIEMPO REAL Y TURNOS

En ocasiones los juegos de estrategia por turnos se vieron obligados a luchar en el mercado con la creciente demanda del género en tiempo real. Aparecieron entonces títulos que daban la posibilidad al jugador de elegir entre una forma de juego y otra. Los títulos más representativos fueron *M.A.X. 2* (Interplay, 1998) y *X-COM: Apocalypse* (Microprose, 1997).



LA BIOGRAFÍA...

SID MEIER

Creador de **Civilization**

Sin duda, un maestro entre maestros. Enseñó su ordenada filosofía de diseño a otros maestros de la talla de Brian Reynolds *(Civilitation II)* o Bruce Shelley *(Age of Empires)*.
A principio de los 80, cofundó la empresa Microprose, líder en el desarrollo de simuladores de vuelos de combate como: *Hellcat Ace* (1983), *F-15 Strike Eagle I* y *II*, *Silent Service*, etc.
Es un gran aficionado a los juegos de estrategia de mesa, y lo reflejó en lo que sería la obra que sentó las bases de este género en ordenadores: *Civilization*. A partir de ahí, todo cambió. Siguieron más títulos como: *Colonization, Sid Meier´s Alpha Centaury, Raildroad Tycoon, Sid Meier´s Civilitation III*, etc.
Abandonó Microprose en 1996 para fundar y dirigir la actual Firaxis con Brian Reynolds y Jeff Briggs. Con 48 años sigue al pie del cañón diseñando y dirigiendo juegos de indiscutible éxito mundial.





1 Warhammer y Steel Panther.



2 UFO de la serie X-COM y X-COM: Apocalypse.



3 Serie Worms.



4 Serie Civilization.

títulos ambientados en la Edad Media como la fantástica serie *WarLords* (Red Orb, 1990-1997). Microsoft y su afán de promover las DirectX y las nuevas posibilidades multimedia de los PCs publicó *Close Combat* (1996), que contribuyó a la aparición de gráficos más realistas y con mejores movimientos en los juegos de estrategia por turnos tradicionales, en los que aún se representaban las celdas de los tableros en pantalla.

En el año 1994, la casa Microprose rompió con el clásico manejo de ejércitos enteros, común en este género, publicando *Ufo* (1994) de su serie *X-COM*, en el que el jugador sólo maneja un grupo de soldados para luchar contra una invasión extraterrestre.

Todos estos títulos se caracterizaban por necesitar pocos recursos para su correcto funcionamiento; aún hoy es posible encontrar juegos de estrategia por turnos en donde los requisitos son mínimos. Sin embargo la llegada del 3D trajo aires de cambio para la representación de las batallas. Ejerció el liderato el juego *Panzer General 3D Assault* (SSI, 2000) y le siguieron otros como *Combat Misión* (Big Time Software, 2000) o *Battle Isle: The Andosia War* (Blue Byte, 2001). Aun así, la perspectiva isométrica y cenital seguían siendo la mejor elección para este tipo de juegos, aún utilizada en el 2000 por Microprose en *X-COM: Genesis*.

Ya en los ordenadores de 8 bits apareció un tipo de juegos de estrategia por turnos muy peculiar que se salía de los cánones establecidos en este tipo de género. Con una representación en 2D y generalmente con vista lateral, consistía en la lucha de un jugador o varios (en el mismo ordenador) con otro/s (humano u ordenador) en un escenario irregular definido aleatoriamente en cada nueva partida. Antes de empezar, cada jugador debía armarse según los créditos que poseía. Básicamente, en cada turno los jugadores estudiaban las estrategias de ataque o defensa según su arsenal. Para PC este tipo de juegos triunfó mundialmente con *Worms* (Team 17, 1995). *Worms* - trasladada a multitud de plataformas como PC, Amiga, PlayStation, Macintosh, etc.- vendió más de un millón de ejemplares y generó una serie de más de una docena de títulos *Worms*: *Worms 2*, *Worms Armageddon*, *Worms World Party*, etc. Con una cinemática fantástica y unos gráficos de dibujos animados, permitía jugar al mismo tiempo hasta 16 jugadores. Además, en la edición *Worms Word Party* es posible jugar a través de internet.

El desarrollo de juegos de estrategia por turnos corría también paralelo a temas que no eran de carácter bélico. Más bien, se acercaban de alguna manera a los simuladores de Dios, ya que el objetivo era prácticamente el mismo: crear y gestionar ciudades, países o civilizaciones enteras. Esta temática de juego en el género de la estrategia por turnos nació con *Civilization* (Micropose, 1992), bastión del género, desarrollado por Sid Meier, que sentó las bases de los juegos de estrategias actuales para ordenador. Con *Civilization* era posible controlar la evolución de una civilización desde cero a partir de un mapa vacío y herramientas para llenarlo. Cada jugador tenía en sus manos el poder de crear su propio mundo y, por turnos, cuidar de él. A *Civilization* siguieron otros títulos de Meier como *Colonization* (1995) y *Civilization II* (1996). Muchos acapararon los estantes siguiendo estos pasos como Sids Meier´s *Alpha Centauri* (Firaxis, 1999), *Imperium Galactica* (Digital Reality, 2000) o Civilization *III* con soporte para internet en *Civilization III: Play the World*.



En el próximo número...

... hablaremos de los juegos de estrategia en tiempo real (RTS).

Cuestionario Videojuegos 10

Preguntas

1. **Describe los pasos para crear el lado de un cuadrado en Blitz3D usando dos triángulos.**

2. **Hemos modelado un hombre, el cual contiene 6 partes: cabeza, tronco, 2 brazos y 2 piernas. ¿Cómo podemos cargarlo desde Blitz3D y texturizar sólo la cabeza?**

3. **¿Cómo podemos definir un FPS (rendimiento) estándar para un juego?**

4. **Para implementar un radar en 2D de un juego en 3D, ¿qué relación existe entre las coordenadas de ambos modos de representación?**

5. **Describe los pasos necesarios para crear un mapeado UV en LithUnWrap de un modelo.**

6. **¿Cómo podemos grabar en disco la textura pintada sobre un modelo en Deep Paint 3D?**

7. **Pasos básicos para realizar un loop en Rebirth.**

8. **Pasos básicos para realizar un loop en Fruity Loops.**

9. **¿Cómo podemos cambiar el volumen de una pista de audio en diferentes puntos en el multipista de Cool Edit Pro 2?**

10. **Una vez mezcladas todas las pistas en el multipista de Cool Edit Pro 2, ¿cómo podemos obtener un solo fichero de audio?**

Respuestas al cuestionario 9

▷ **1.** Escalando la entidad terreno como sigue:
ScaleEntity terreno,10,300,10

▷ **2.** Cargamos, por ejemplo, dos texturas:
textura1=LoadTexture("textura_suelo1.png")
textura2=LoadTexture("textura_suelo2.png")
Escalar la textura 1 y aplicarla al terreno:
ScaleTexture textura1,512,512; Tamaño de la textura 512X512
EntityTexture Terreno,textura1,0,0
Escalar la textura 2 y aplicarla al terreno:
ScaleTexture textura1,128,128; Tileado (entre 2 hasta 512 siempre Múltiplo de 2)
EntityTexture Terreno,textura2,0,1

▷ **3.**
```
Print "Raiz cuadrada de 25 es: "+Raiz_Cuadrada(25)
End
Function Raiz_Cuadrada(numero)
    Return Sqr (numero)
End Function
```

▷ **4.** En primer lugar hay que crear y situar una cámara para ver el entorno 3D
Camara=CreateCamara()
CameraRange Camara,0.1,Rango_Max_Vision
Seguidamente hay que iluminar el entorno:
AmbientLight 150,150,150
Sol=CreateLight()
Y para finalizar crear, texturizar y colocar el terreno:
Terreno=LoadTerrain("Terreno1.png"); Mapa de alturas de 512x512
ScaleEntity Terreno,10,300,10; Por ejemplo
TerrainDetail Terreno,Num_polígonos,True
Textura=LoadTexture("Textura_terreno1.png")
ScaleTexture Textura, 512, 512
EntityTexture Terreno,Textura
PositionEntity Terreno,X#, Y#, Z#

▷ **5.** (A) Crear el esqueleto a partir de uniones.
(B) Asignar los vértices del modelo a las uniones.
(C) Mover las uniones y crear keyframes.

▷ **6.** Las animaciones se crean en el *Keyframer* en el modo *Animation* y deben ir entre *KeyFrames* a lo largo de la línea de tiempo.

▷ **7.** El ruido lo podemos solucionar eliminándolo o reduciéndolo aplicando el efecto *Noise Reduction*.

▷ **8.** En GoldWave la calidad de un sonido se puede cambiar realizando un resampleo con la función *Resample* en *Effects*.

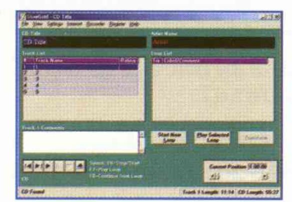
▷ **9.** Un cambio del volumen en Cool Edit Pro significa modificar la envolvente del sonido por medio de la creación de uno nuevo a través de la función *Create Envelope* situada en *Effects \ Amplitude \ Envelope*.

▷ **10.** (A) Crear un Script nuevo y darle un nombre al archivo en la ventana *Scripts and Batch Processing*.
(B) Añadir acciones al nuevo archivo por medio del botón "Record".
(C) Una vez grabadas las acciones, hay que registrar el nuevo script pulsando en el botón "<<Add to Collection>>".

Contenido CD-ROM 10

▶ AUDIO

■ Slow Gold 7.1



Cambia el tempo de tus composiciones sin que éstas pierdan su ritmo.

■ Advanced MP3/WMA Recorder 3.2.6

Grabación de audio en cualquier formato y en modo streaming.

■ Alcohol 1.2.3

Útil herramienta para realizar backups de CDS de audio, para que no pierdas nada de lo que compongas.

■ Blaze Media Pro 3.01

Edita el sonido y conviértelo al formato que desees.

■ Fruity Loops 3.3.0

Nueva oportunidad para conseguir la demo de este maravilloso programa.

■ Rebirth

También te ofrecemos de nuevo la ocasión de hacerte con esta excelente aplicación.

▶ DISEÑO 2D

■ Picasa 1.0.1



Organiza automáticamente todas tus imágenes con este potente clasificador.

■ Document Imaging Application 2.1

Sencilla aplicación para adquirir imágenes y archivarlas.

■ EyeBatch 2.0.12

Aplica múltiples y complejos efectos a tus imágenes.

■ Ashampoo Illuminator 1.50

Clasifica, visualiza y ten ordenadas todas tus imágenes fácilmente.

■ Fractal Fantasy 1.5

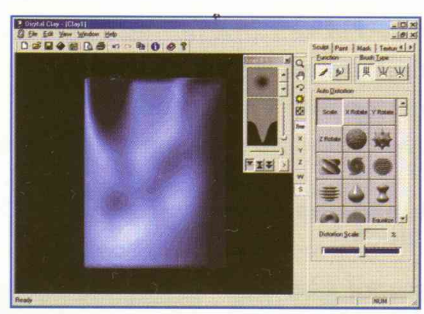
Crea fractales realistas o artísticos para dibujar árboles, nubes, paisajes, etc...

■ IMatch 3 3.1

Editor de imágenes muy profesional que te permitirá obtener imágenes con una calidad excepcional.

▶ DISEÑO 3D

■ Digital Clay 1.60



Diseña formas en tres dimensiones de un modo sencillo y muy intuitivo.

■ EBook 3D Wizard 4.6

Con esta aplicación podrás crear las imágenes en 3D para la portada del CD del juego.

■ Infinity Textures 2.3

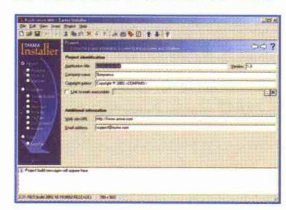
Crea un montón de texturas para tus modelos en 3D.

■ 3D Studio

Demo del célebre programa de creación de modelos en 3D.

▶ PROGRAMACIÓN

■ Tarma Installer 2.0



Crea un instalador para tu programa fácil y rápidamente.

■ Soloway Controls 1.0

Diseña fácilmente las pantallas del juego gracias a este utilidad.

■ Visual WinHelp 1.0

Con esta aplicación podrás crear ventanas de ayuda de Windows, lo que te será muy útil para tu juego.

■ Tile Studio 2.33

Programa muy útil para crear tiles y sprites así como un editor de mapas.

■ Spriteworks 1.0

Original librería de gráficos para sprites que podrás usar en tus porgramas.

▶ JUEGOS

■ Civilization 2: Test of time

Segunda parte del mítico juego de la saga Civilization, bandera en los juegos de estrategias por turnos no bélicos.

■ Panzer General 3D

Uno de los primeros juegos de estrategia por turnos que usó las tres dimensiones.

■ Warhammer



El clásico juego estratégico de tablero, en su versión para PC.

■ Worms World Party

Excelente juego, secuela del mítico Worms, que tantos adeptos tuvo.

■ Zone of Fighters

Como todas las semanas, nuestro juego.

▶ VÍDEO

■ FX Movie Joiner 4.8.2

Con esta interesante utilidad podrás mezclar sin problemas archivos de vídeo con imágenes.

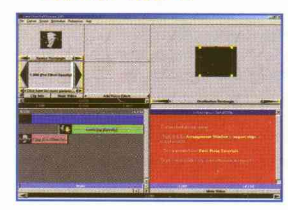
■ Honestech MPEG Editor 3.0

Robusto editor de vídeo que te será de gran ayuda.

■ Random Frame 1.2

Extrae los frames que desees de tus vídeos.

■ Zwei-Stein Video Editor 3.01.



Más de 64 efectos que podrás añadirle a tus vídeos para darles un toque muy personal.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.